Data compression algorithms for flow tables in Network Processor (RuNPU).

Nikita Nikiforov
Lomonosov Moscow State University
Moscow, Russia
nickiforov.nik@gmail.com

Dmitry Volkanov
Lomonosov Moscow State University
Moscow, Russia
volkanov@asvk.cs.msu.ru

***Abstract*—This paper addresses the problem of packet classification within a network processor (NP) architecture without the separate associative device. By the classification, we mean the process of identifying a packet by the header. The classification stage requires the implementation of data structures to store the flow tables. In our work we consider NP without the associative memory. In considering NP flow tables represent like an assembly language program. For translating flow tables into assembly language programs, tables translator was used. Nowadays flow tables can take tens of megabytes of memory. This is the reason for implement data compression algorithms in flow table translator. In this work we provide the data compression algorithms: Optimal rule caching, recursive end-point cutting and bit string compression. An evaluation of the implemented data compression algorithms was performed on a simulation model of the NP.**

***Index Terms*—Network processor, software-defined networks, packet classification, data compression.**

I. Introduction

At present, software-defined networks (SDN) are in active developing and require high-performance switches. The main functional element of the high-performance SDN switch is programmable network processor. The network processor is system-on-chip specialized for network packet processing. In our work we consider the programmable NP. By programmable we mean such NP, that supports to change the packet processing program and the set of processed header fields on the fly.

This article will discuss about data compression algorithms used for flow tables. Flow tables needs for packet classification process. Flow table is the set of flows, that defines by OpenFlow protocol. Each rule contains match field, bit string by witch a packet can be identified and set of actions, that NP performs on this packet. The classification is the process of identification a network packet by it's header.

II. Network processor architecture

In considered NP the pipeline architecture is used, each pipeline consists of 10 computing blocks. To avoid complex organization of memory, there is no associative memory in the considered NP. The NP uses the same memory for commands and data.

Let consider the pipeline NP architecture. Each computing block has access to the memory area where the program with data are located. There is a limit on the number of clock cycles that one packet can process on a processing block, it corresponds to 25 clock cycles. All memory to store assembly language program that represents flow tables is 512 kilobytes. Due to the instruction set architecture, there is no separate memory area where data is stored. Therefore, the microcode contains all the data, required to classify packets.

A. Flow tables translator

Flow table translator is a tool that performs on CPU. It used for flow table translating into assembly language program, that can by interprited by NP.

III. The problem

Let consider OpenFlow tables formalisation. Ordered set of all considered attributes we'll denote as $I = \{m_1, m_2, \dots, m_k\}$. Every attribute m_i from the set I described by bit string $m_i \in \{0, 1, *\}_i^W$. In this article symbol $*$ denote any bit. But, if $\exists m_i^j \in m_i$ and $m_i^j = *$, then for $\forall m_i^k$, where $k > j$, and $m_i^k = *$. Length of the attribute will be $len(m_i) = W_i$.

Flow table will be represented as set of rules $R = \{r_1, r_2, \dots, r_n\}$. With every rule r_i bind the features:

- Index i;
- Priority $p_i \in Z_+$;
- Vector values of the attributes $f_i = \{f_i^1, f_i^2, \dots, f_i^k\}$, there f_i^j is attribute value $m_j \in I$.
- Set of actions $A_i = \{a_1, a_2, \dots, a_z\}$.

Network packet header x and its metadata with vector values of the attributes $g = \{g^1, g^2, \dots, g^k\}$ $(x \rightarrow g)$, match rule $r_i \in R$ with vector values of the attributes $f_i = \{f_i^1, f_i^2, \dots, f_i^k\}$ and priority p_i (Rule $r_i \in R$ identifies network packer with vector values of the attributes g), if:

1) vector values of the attributes g match vector values of the attributes f_i, $\forall g_i \in g$, $len(g_i) = len(f_i)$. $\forall f_i^{lj} \in f_i^l$, $f_i^{lj} \in \{*, g^{lj}\}$, $l = \overline{1, k}$;
2) the priority p_i is the highest among all rules $r_j \in R$, witch vector g match vectorf_j.

The set of rules R must satisfy the following constraint. For any two rules $r_i, r_j \in R, r_i \neq r_j$, if their vectors of values intersect, there is a set of attribute values. This set corresponds to vectors of values of attributes of both rules $p_i \neq p_j$.

Let's introduce the function for network packet identification $x \to g$in flow table R, (will be note as $R(x)$). It will return a set of actions, that corresponds the rule $x \to g$.

$R(x) = A_{r_i}$, there A_{r_i} is the set of actions $r_i \in R$.

We need introduce similar concept of the sets of rules R_1 and R_2. The set R_1 is similar to the set R_2, when for any network packet header, that can be identified by some rule from the set $r_i \in R_1$. And for this network packet header exist another rule that identifies it $r_j \in R_2$, and $A_i = A_j$.

We need to develop algorithm for compressing flow tables. This algorithm must translate input flow table (the set of rules R_1 into new compressed set of rules R_2.

1) The set of rules R_1 is similar to the set of rules R_2.
2) Power of the set R_2 must be lower than power of the set R_1.

IV. Related work

In this section we'll take a view on data compression algorithms, that already used for other network processors. We need to choose algorithms for use in our NP. For choosing algorithms we implemented criteria.

1) Compression rate, needs for algorithm performance evaluation.
2) Evaluation of compression algorithm complexity.
3) Usability compressed flow tables without decompression.
4) The need to use external memory by the algorithm.

A. Most common data compression algorithms

Data compression algorithms have evolved over the years. Nowadays compression algorithms can be used in many different ways. In this section we'll take a view on the algorithms that compressed data in binary format. There are most known of them:

- Huffman codding,
- JPEG,
- LWZ,
- zip.

These algorithms needs decompression for data usage. And this is why we'll not use them in our flow table translator.

B. Optimal rule caching

This is more specify data compressing algorithm. It used for table compressing in SND switches. It's based on search tree structure, that builds based on rules usage frequencies. There are two trees. The first tree - most usable rules. This tree translates into assembly language program. The second tree - another rules, it stores in CPU memory.

C. Recursive end point cutting

This algorithm based on HyperSplit tree usage. Compressing performs by destroying duplication rules [1]. This algorithm allows operations with flow table without dull rebuilding tree.

By rules duplication we'll understand next rules:

- The rule, that contains in node duplicates by rule in leaf node. (particle duplication).
- The rule, that contains in node duplicates by rules in all leafs nodes. (full duplicating rule).

V. Our solution

Введём операцию последнего значащего бита признака $last(m_i) = j$, такое, что $m_i^j \in \{0,1\}$ и $m_i^{(j+1)} = *$. Назовём правила $r_i \in R$ и $r_j \in R$ похожими, если для $\forall u \in len(f_i)$ верно, что $last(f_i^u) = last(f_j^u) = l$, при этом $f_i^{ul} \neq f_j^{ul}$, и $A_i = A_j$. Для описания данного алгоритма потребуется ввести дополнительные обозначения. Введём понятие распределения заголовков пакетов P, где p_x обозначает вероятность получения пакета $x \to g = \{g^1, g^2, \ldots, g^k\}$. Также введём понятие коэффициента правильности $T_P(R_1, R_2)$, где R_1 и R_2 две различные таблицы потоков. Таким образом коэффициент правильности обозначает вероятность того, что заголовок пакета, согласно распределению P, будет в идентифицироваться правилами $r_1 \in R_1$ и $r_2 \in R_2$ и их наборы действий совпадают $A_1 = A_2, A_1 \in r_1, A_2 \in r_2$.

$$T_P(R_1, R_2) = \sum_{x \to g, R_1(x) = R_2(x)} p_x$$

Введём оптимальное значение коэффициента правильности для заданной таблицы потоков R, числа правил n и распределения заголовков P.

$$\zeta(n, R, P) = \max_{R_i, |R_i| <= n} T_P(R, R_i)$$

Таким образом алгоритму необходимо найти и построить таблицу потоков R_a, основанную на данной таблице потоков R с наименьшим количеством правил n_0 и максимальные оптимальным коэффициентом правильности $\zeta(n, R, P)$. Пусть p^i популярность (вероятность) выбора правила $r_i \in R$, в соответствие с распределением заголовков P. Пусть правила в таблице потоков R расположены в порядке не возрастания их популярности. Тогда:

$$\zeta(n, R, P) \geq \sum_{i \in [1,n]} p^i + 1 - \sum_{i \in [1,n_0]} p^i \geq n/n_0$$

VI. Evaluation

Для оценки параметров необходимо исследовать программу на языке ассемблера, получаемую при использовании системы трансляции с алгоритмами сжатия и без. Для каждой программы, с помощью эмулятора сетевого процессорного устройства, будут исследоваться следующие параметры:

- Объём памяти занимаемой программой при обработке пакетов на эмуляторе сетевого процессорного устройства.
- Среднее время обработки пакета в тактах сетевого процессорного устройства.

Для проведения экспериментального исследования, необходимо последовательно выполнять следующие действия для каждого набора входных данных:

1) Выбрать таблицу потоков для данного эксперимента.
2) Провести трансляцию выбранной таблицы потоков в программу на языке ассемблера:
   – без использования алгоритмов сжатия, обычное дерево;
   – без использования алгоритмов сжатия, с АВЛ деревом;
   – с использованием разработанных алгоритмов сжатия.
3) Провести эмуляцию работы сетевого процессорного устройства с полученными программами на языке ассемблера.
4) Провести оценку результатов полученных в данном эксперименте.

VII. Future work

References

[1] Yeim-Kuan Chang and Han-Chen Chen. “Fast packet classification using recursive endpoint-cutting and bucket compression on FPGA”. In: *The Computer Journal* 62.2 (2019), pp. 198–214.